

Dynamic Self-Aware Runtime Software for Exascale Systems

Christian Engelmann*, Geoffroy Vallée, and Thomas Naughton

Oak Ridge National Laboratory

* (865) 574-3132 / engelmann@computer.org

Prof. Frank Mueller

North Carolina State University

(919) 515-7889 / mueller@cs.ncsu.edu

At exascale, the power consumption, resilience, and load balancing constraints, especially their dynamic nature and interdependence, and the scale of the system require a radical change in future high-performance computing (HPC) operating systems and runtimes (OS/Rs). In contrast to the existing static OS/R solutions, an exascale OS/R is needed that is aware of the dynamically changing resources, constraints, and application needs, and that is able to autonomously coordinate (sometimes conflicting) responses to different changes in the system, simultaneously and at scale. **To provide awareness and autonomic management, a novel, scalable and self-aware OS/R is needed that becomes the “brains” of the entire X-stack (Figure 1).** It dynamically analyzes past, current, and future system status and application needs. *It optimizes system usage by scheduling, migrating, and restarting tasks within and across nodes as needed to deal with multi-dimensional constraints, such as power consumption, permanent and transient faults, resource degradation, heterogeneity, data locality, and load balance.*

Interaction with other OS/R components, the programming model, and the application are performed in a control loop through (1) awareness APIs offering a holistic view of the entire current system state, (2) a unified node-local controller processing (a) the entire current system state, (b) quality of service (QoS) requests identifying future needs, and (b) models deciding on corrective actions, and (3) feedback APIs to perform corrective actions if needed. While models for power management, resilience, and load balancing define the general control loop behavior, QoS requests issued by the application define the control loop’s policies and steer the employed mechanisms. While the first order of business for the control loop is to do no harm, it incrementally optimizes the system. However, a fully optimal solution across all nodes is likely unachievable due to dynamic behavior, scale, and real-time requirements.

There are three classes of awareness: (1) the dynamically changing resources, including availability, reliability, and performance, (2) the present use of these resources, including task allocation, data locality, utilization, and power consumption, and (3) the future needs from these resources, including task schedule, multi-application workflow, and QoS requests. The real-time control processes awareness data using models to provide self-awareness with optimization at discrete time intervals. Feedback control is used for observed deviations and feed-forward is used for predicted deviations. Competing models interact through feedback, cooperating models interact via feed-forward, and unified models combine capabilities. Each node performs a local control optimizing its usage. For scalability, information across nodes is disseminated using gossip protocols [Dim08] and enclave topologies, piggybacking node-level awareness data on application messages whenever possible. The models are able to process data from the same interval within an enclave and from past intervals coming from other parts of the system.

The work for offering awareness needs to focus on (1) identifying the required input data (based on the decision models) and its properties (e.g., linear or non-linear), (2) designing the awareness API to collect the required input data, and (3) creating QoS request templates to offer basic initial capabilities (e.g. simple node-local power-aware task scheduling/migration, reliability-aware task scheduling/migration, and application-directed load balancing). The effort for providing self-awareness has to aim at (1) creating decision models based on an architecture-aware, parameterized resource abstraction (e.g., per-resource power consumption, reliability, performance, and utilization), (2) designing the feedback API to communicate corrective actions (e.g. change task schedule, migrate tasks, and restart tasks), (3) identifying the control mechanism (e.g., proportional-integral-derivative or machine learning), and (4) investigating the control loop’s stability (e.g. deriving the transfer function). The work for offering a self-aware runtime prototype must target a library-based event-driven approach that can be integrated with any exascale execution model (e.g., MPI+X and HPX [Hel12]) and directly with applications. The research further has to include evaluating the effectiveness of the self-aware runtime prototype with DOE applications on Leadership systems.

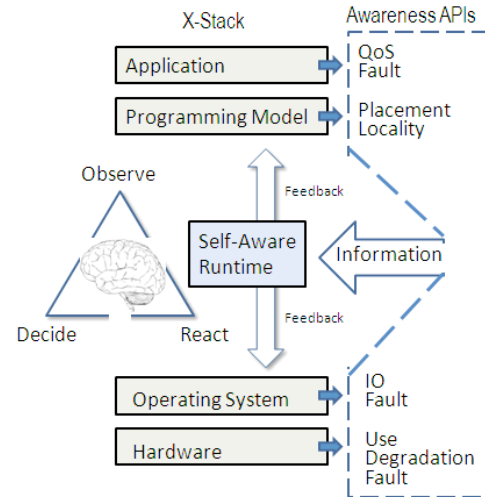


Figure 1: The dynamic, self-aware runtime forms the “brains” of the X-stack using awareness APIs to multiple X-stack components

Related Work

Power-aware HPC has focused on (1) statically designing energy efficient systems [Bel11], (2) trading off performance for energy savings using dynamic voltage/frequency scaling [Zhu07], and (3) optimizing performance-per-Watt through power-aware scheduling [Li11]. There is currently no power-aware HPC solution for extreme-scale systems. Also, instead of optimizing energy efficiency, maximizing performance with a given power ceiling is needed at exascale. Resilience in HPC includes (1) reactive mechanisms, like application- and system-level checkpoint/restart [Wan10] and system-level message logging [Lem04], (2) proactive approaches, such as (a) system-level migration of tasks in anticipation of faults [Wan12], (b) reliability-aware scheduling [Got07], and (c) rejuvenation [Nak10], (3) programming model approaches, like FT-MPI [Fag05] and containment domains [Sul11], and (4) algorithm-based fault tolerance for (a) dense linear algebra kernels [Du12a], (b) the LINPACK benchmark [Dav11], and (c) linear hyperbolic and parabolic PDE solvers [Lta08]. While these solutions address dynamically changing systems, they are often limited in scaling due to centralized decisions and global actions. Load balancing has been realized via (1) data repartitioning embedded in the application [Lan01], (2) data repartitioning supported by a data management library [Dev02], or (3) core oversubscription and task migration offered by a runtime [Bha02]. Similar to the resilience solutions, scalability is a problem. For all three, existing solutions do not coordinate with other components and do not consider other dynamic properties.

Assessment

Challenges addressed: This work addresses the exascale OS/R challenges of power, resilience, and dynamic environments. By 2020, an exascale system may have up to 1 million nodes with up to 1 billion cores in total using 7 nm process technology operating at near-threshold voltage and the entire system bound by a power envelope of 20 MW [Kog08, Dal12, Kau12]. This poses several challenges for power, resilience, and performance. Present OS/Rs have almost no awareness of changes to system resources during program execution. However, future systems need to deal with a variety of dynamic changes. As a system's self protection mechanism may automatically throttle or shutdown resources to avoid exceeding a node's or the system's power budget, power-unaware resource usage results in performance degradation and resource outages. Permanent and transient faults will occur continuously due to decreased component reliability and increased component counts. At 1 billion cores, even a tiny amount of load imbalance due to static task scheduling and unforeseen resource contention severely affects overall performance. The self-aware OS/R addresses these challenges in a holistic approach using dynamic adaptation.

Maturity: Prior work has shown that power consumption, resilience, and load balancing can be individually dynamically optimized. This work extends these efforts and aims at a holistic solution supporting extreme-scale by leveraging existing technology, such as gossip protocols [Dim08], dynamic real-time runtime environments [DiS07], and scalable communication infrastructures [CCI12]. The result is a self-aware OS/R that addresses the exascale challenges of power, resilience, and dynamic environments.

Uniqueness: The approach aims at OS/R self-awareness with scalable, autonomic management using a control loop guided by power, resilience and load balancing models and application QoS requests. It addresses critical challenges that are exclusive to exascale systems, such as maximizing performance within a power ceiling, dealing with continuous faults, and dynamic load balancing, all at extreme scale.

Novelty: For all three, power-aware computing, resilience, and load balancing, existing implementations are standalone solutions that do not coordinate with the other components of the software stack and that do not consider the other dynamic system properties. The proposed concept provides a radically different OS/R with self-awareness capabilities that do not exist today. The targeted holistic approach of combining dynamic optimization for power, resilience, and performance at scale across the stack is new.

Applicability: This work deals with dynamically changing resources, i.e., changing resource availability, reliability, and performance, and with design constraints, i.e., fixed system- and subsystem-wide power, by dynamically adapting resource utilization in concert with the entire system and the running application. It is applicable to other areas with similar issues, such as cloud and mobile computing.

Effort: The described concept represents high-risk/high-payoff R&D in HPC system software and in modeling the power consumption, resilience, and load balancing constraints of extreme-scale HPC systems and applications. A large part is the fundamental research required to make the revolutionary leap to put real-time awareness and decision making into a runtime. A prototype self-aware runtime needs to be created and evaluated using DOE applications, first using only node-local control and then with scalable, enclave-based control. A testing methodology to quantify its effectiveness needs to be developed, as well as, corresponding experiments have to be performed on DOE's Leadership computing systems.

References

- [Bel11] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Y. Zomaya. A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems. *Advances in Computers*, 82:47-111, 2011.
- [Bha02] M. A. Bhandarkar, L. V. Kalé, E. d. Sturler, and J. Hoeflinger. Adaptive Load Balancing for MPI Programs. 1st International Conference on Computational Science (ICCS), Part I, pp. 108-117, 2001.
- [Cat07] U. Catalyurek, E. Boman, K. Devine, D. Bozdag, R. Heaphy, and L. A. Riesen. Hypergraph-based Dynamic Load Balancing for Adaptive Scientific Computations. 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2007.
- [CCI12] CCI: Common Communication Interface - A simple, portable, high-performance, scalable, and robust communication interface for HPC and Data Centers. At <http://cci-forum.com>, 2012.
- [Dal12] J. Daly et al.. Inter-Agency Workshop on HPC Resilience at Extreme Scale. 2012.
- [Dav11] T. Davies, C. Karlsson, H. Liu, C. Ding, and Z. Chen. High Performance Linpack Benchmark: A Fault Tolerant Implementation without Checkpointing. 25th ACM International Conference on Supercomputing (ICS), 2011.
- [Dev02] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan Data Management Services for Parallel Dynamic Applications. *Computing in Science and Engineering*, 4(2):90-97, 2002.
- [Dim08] A. Dimakis, A. Sarwate, M. Wainwright. Geographic Gossip: Efficient Averaging for Sensor Networks. *IEEE Transactions on Signal Processing* 56(3):1205-1216, 2008.
- [DiS07] E. Di Saverio, M. Cesati, C. Di Biagio, G. Pennella, and C. Engelmann. Distributed Real-Time Computing with Harness. 14th European PVM/MPI Users' Group Meeting (EuroPVM/MPI), pp. 281-288, 2007.
- [Du12] P. Du, A. Bouteiller, G. Bosilca, T. Herault, J. Dongarra. Algorithm-Based Fault Tolerance for Dense Matrix Factorization. 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP) 2012, pp. 225-234, 2012.
- [Fag05] G. Fagg, T. Angskun, G. Bosilca, J. Pjesivac-Grbovic, J. Dongarra. Scalable Fault Tolerant MPI: Extending the Recovery Algorithm. In *Proceedings of 12th European Parallel Virtual Machine and Message Passing Interface Conference (Euro PVM/MPI)*, pp. 67, 2005.
- [Got07] N. R. Gottumukkala, C. Leangsuksun, N. Taerat, R. Nassar, and S. L. Scott. Reliability-aware Resource Allocation in HPC Systems. *IEEE International Conference on Cluster Computing*, pp. 312-321, 2007.
- [Hel12] T. Heller, H. Kaiser and K. Iglberger. Application of the ParalleX Execution Model to Stencil-based Problems. *International Supercomputing Conference (ICS)*, 2012.
- [Kau12] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar. Near-threshold voltage (NTV) design: Opportunities and challenges. 49th Annual Design Automation Conference (DAC), pp. 1153-1158, 2012.
- [Kog08] P. Kogge et al.. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, 2008.
- [Lem04] P. Lemarinier, A. Bouteiller, T. Herault, G. Krawezik, and F. Cappello. Improved Message logging versus Improved Coordinated Checkpointing for Fault Tolerant MPI. 6th IEEE International Conference on Cluster Computing (Cluster), pp. 115-124, 2004.
- [Li11] K. Li. Design and Analysis of Heuristic Algorithms for Power-Aware Scheduling of Precedence Constrained Tasks. 24th IEEE International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), pp. 804-813, 2011.
- [Lta08] H. Ltaief, E. Gabriel, and M. Garbey. Fault Tolerant Algorithms for Heat Transfer Problems. *Journal of Parallel and Distributed Computing*, 68(5):663-677, 2008.
- [Nak10] N. Naksinehaboon, N. Taerat, C. Leangsuksun, C. Chandler, and S. L. Scott. Benefits of Software Rejuvenation on HPC Systems. 8th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 499-506, 2010.
- [Sul11] M. Sullivan, D.H. Yoon, and M. Erez. Containment Domains: A Full-System Approach to Computational Resiliency. Technical report TR-LPH-2011-001, The University of Texas at Austin, 2011.
- [Wan10] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Hybrid Checkpointing for MPI Jobs in HPC Environments. 16th IEEE International Conference on Parallel and Distributed Systems, pp. 524-533, 2010.
- [Wan12] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Proactive Process-Level Live Migration and Back Migration in HPC Environments. *Journal of Parallel and Distributed Computing*, 72(2):254-267 2012.
- [Zhu07] Y. Zhu and F. Mueller. Exploiting Synchronous and Asynchronous DVS for Feedback EDF Scheduling on an Embedded Platform. *ACM Transactions on Embedded Computing Systems*, 7(1):26, 2007.